Finding Safe Policies in Model-Based Active Learning

David Martínez, Guillem Alenyà and Carme Torras

***Abstract*— Task learning in robotics is a time-consuming process, and model-based reinforcement learning algorithms have been proposed to learn with just a small amount of experiences. However, reducing the number of experiences used to learn implies that the algorithm may overlook crucial actions required to get an optimal behavior. For example, a robot may learn simple policies that have a high risk of not reaching the goal because they often fall into dead-ends.**

We propose a new method that allows the robot to reason about dead-ends and their causes. Analyzing its current model and experiences, the robot will hypothesize the possible causes for the dead-end, and identify the actions that may cause it, marking them as dangerous. Afterwards, whenever a dangerous action is included into a plan which has a high risk of leading to a dead-end, the special action *request_teacher_confirmation* will be triggered by the robot to actively confirm with a teacher that the planned risky action should be executed.

This method permits learning safer policies with the addition of just a few teacher demonstration requests. Experimental validation of the approach is provided in two different scenarios: a robotic assembly task and a domain from the international planning competition. Our approach gets success ratios very close to 1 in problems where previous approaches had high probabilities of reaching dead-ends.

I. INTRODUCTION

Many robotic applications include dangerous actions that, if applied under certain circumstances, may yield irrecoverable problems that we call dead-ends. Robots should learn to identify these dangerous actions and be specially careful before executing them in order to avoid these dead-ends. This is a very challenging problem when learning tasks, as models not yet completely learned may lack important constraints and effects needed to yield safer plans that avoid dead-ends. In this paper we propose a new method that extends active learning approaches to identify dangerous actions and avoid repeatedly falling in the same dead-ends.

For example, this method can be applied to the robotic assembly task shown in Fig. 1. In this task, the robot has to learn and execute simple manipulation actions, like placing a peg into a hole, and repeat them until the assembly is completed. A common problem is that parts not grasped correctly may fall. This is important specially for round pegs which often roll away from the reachable workspace of the robot when they fall. The objective is to enable the robot to identify the failure causes, and let it request help to a teacher when it cannot find good alternatives to avoid the problem. The teacher can show to the robot, for example, a new grasping strategy for certain parts.

This work was supported by EU Project IntellAct FP7-ICT2009-6-269959, by CSIC project MANIPlus 201350E102 and by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

Authors are with Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Llorens i Artigas 4-6, 08028 Barcelona, Spain {dmartinez,galenya,ctorras}@iri.upc.edu

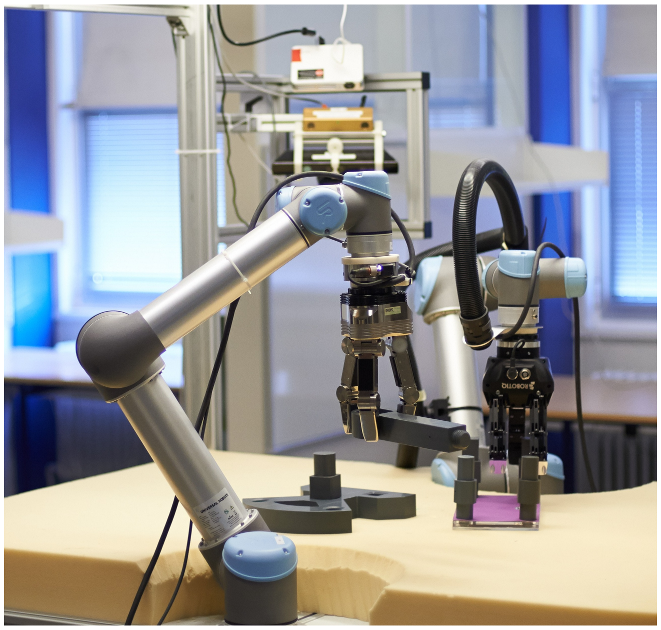


Fig. 1. Marvin platform assembling the Cranfield benchmark.

All the required actions to complete a task are codified using rules. These rules, including the ones that account for the risk of a dead-end, are used by a decision making module to produce valid plans by analyzing the current state of the robot and its environment. It selects which actions should be executed to complete the task, and asks for interaction with the human teacher whenever it is necessary. The decision maker starts with no previous knowledge about the actions that can be executed, and learns them incrementally as it gets experiences from teacher demonstrations and action executions.

The decision maker uses model-based Reinforcement Learning (RL), which is a common paradigm for learning tasks in robotics [1]. The robot has to execute actions to learn the model that represents the scenario and to complete the requested tasks. Thus, the robot needs to face the exploration-exploitation dilemma [2], that consists in choosing whether to explore (execute actions to learn unknown parts of the model that would lead to better plans in the long term) or to exploit (select actions to complete a task with maximum reward). Moreover, increased performance and versatility can be obtained through the combination with Learning from Demonstration (LfD) [3], [4]. LfD permits requesting help from a teacher when no solution has been found and a lot of exploration would be needed to obtain one.

The combination of RL, LfD and relational generalizations between similar objects [4], [5], permits reducing a lot the number of exploration actions with the drawback that some parts of the domain may remain unexplored. This is a problem in some domains where dead-ends are common. To overcome repeating these failures, the robot will have to detect the causes of the dead-ends and learn how to avoid them. In this paper we propose a method that allows the robot to learn how to avoid these dead-ends by requesting a few extra teacher demonstrations.

There have been some proposals on diagnosing plan executions by identifying the agent or action that make the plan fail [6]. Other approaches monitor the environment and the actual effects of executed actions, to observe whether something goes wrong, and then re-plan or repair the plan if needed [7], [8]. In [9] a learner is included into a robust planning framework to adapt plans based on previous experiences about the actions. These works assume that the model used for planning is already known, but something has changed or happened that makes the plans fail. In our case we are in a different situation as we are learning the model: some parts of the model that are still unknown may be required to find plans that avoid dead-ends. Therefore previous approaches find difficulties that weren't supposed to happen, while our approach identifies the parts of the model that are problematic, and tries to learn them better to figure out how to avoid dead-ends.

A general method to analyze dead-end causes is using planning *excuses* [10], [11], which are state changes that would have to be made to find a proper plan. In this work we propose to use these planning excuses to detect dangerous actions that can lead to dead-ends before they are executed. Once a dangerous action has been identified, if a plan includes it, the system will issue a teacher interaction request to confirm that there isn't a better alternative.

To summarize, we propose a new method to avoid dead-ends when using reinforcement learning with strong generalizations that make the robot learn very fast at the cost of leaving large parts of the domain unexplored. Unlike other approaches, our method avoids dead-ends when having models that are not completely learned yet. Moreover, the robot is the one that actively interacts with the teacher to learn how to overcome the dead-ends, contrarily to other approaches that require the teacher to continuously monitor the robot behavior.

The paper is organized as follows. After this introduction, the background needed to understand our proposal is presented in Section II. Afterwards, the method to avoid dead-ends is detailed in Section III. Section IV explains the experiments conducted in a simulator and a real robot, and the obtained results are shown. Finally some conclusions and future work is presented in Section V.

II. The Decision Making and Learning Algorithm

The initial assumption is that the perception modules can provide full observability, and that robot actions are uncertain because they may fail or yield unexpected outcomes. Therefore, **Markov Decision Processes** (MDP) can be used to formulate fully-observable problems with uncertainty. A finite MDP is a five-tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \alpha \rangle$ where $\mathcal{S}$ is a set of possible discrete states, $\mathcal{A}$ is the set of actions that the robot can execute, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $\alpha \in [0, 1)$ is the discount factor. The goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ which chooses the best action for each state to maximize future rewards. To that end, we have to maximize the value function $V^\pi(s) = E[\sum_t \alpha^t R(s_t, a_t) | s_0 = s, \pi]$ that is the sum of expected rewards.

The decision maker uses a symbolic representation to represent the model and the states. A state is composed of a set of symbolic predicates that represent the scenario that the robot is interacting with. The model consists of a set of **Noisy Indeterministic Deictic (NID)** rules [12]. The transition model T is represented with a set of NID rules Γ which are defined as:

$$a_r(\chi) : \phi_r(\chi) \to \begin{cases} p_{r,1} \ : \ \Omega_{r,1}(\chi) \\ \quad \vdots \\ p_{r,n_r} \!: \ \Omega_{r,n_r}(\chi) \\ p_{r,0} \ : \ \Omega_{r,0} \end{cases}, \qquad (1)$$

where a_r is the action that the rule represents, $\phi_r(\chi)$ are the preconditions for the rule to be applicable, $\Omega_{r,i}$ are the effects defining the set of predicates that are changed in the state with probability $p_{r,i}$ when the rule is applied, $\Omega_{r,0}$ is the noisy effect that represents all other, unmodeled, rare and complex effects, and χ is the set of variables of the rule. A NID rule represents one action, while each action may be represented by several rules. Each state-action pair (s, a) is covered by just one rule r as all the rules defining one action have disjoint preconditions $\phi_{r_{a,i}} \wedge \phi_{r_{a,j}} = \emptyset \mid \forall i, j$.

A reinforcement learning problem consists in learning a transition model T which is unknown a priori. A RL algorithm has to balance exploration (try different actions to increase the system knowledge and obtain better policies in the long term) and exploitation (choose actions to maximize the reward according to the current policy) to obtain good results. In particular we are using model-based RL where a model is estimated from experiences, and this model is then used to plan the actions that the system executes. We are using Glutton [13] as a symbolic planner and Pasula et al.'s learning system [12] within the RL algorithm.

Learning from Demonstration is a supervised learning method where a teacher shows a system how to perform an action or a task. In our problem the teacher will tell the robot how to perform an action, its symbolic name and its parameters. We consider that the teacher chooses actions using the optimal policy π^*.

The decision maker uses the **REX-D** [4] algorithm that combines RL and LfD to learn scenarios with fewer exploration actions that in other approaches, while not requiring many teacher demonstrations either. It has to balance exploration, exploitation and teacher demonstrations. To that end, REX-D explores the state-action pairs considered unknown before exploiting the model to try to obtain the maximum reward. Moreover, if no solution is found in a known state, then REX-D requests a teacher demonstration as it considers that a completely new action or yet unknown effects of an action under different preconditions need to be demonstrated. Teacher demonstrations are actively requested by the decision maker, so no human monitoring is required.

REX-D uses relational representations [5], generalizing over different objects of the same type as they exhibit the same behavior. A context-based density count function [5] is used to handle the exploration-exploitation dilemma, which reduces the number of samples before considering states as known by grouping them in contexts:

$$k(s,a) = \sum_{r \in \Gamma} |E(r)| I(r = r_{s,a}), \tag{2}$$

where $|E(r)|$ is the number of experiences that cover the rule with any grounding, and $I()$ is a function that takes the value 1 if the argument evaluates to true and 0 otherwise. State-action pairs which have been explored less times than a fixed threshold $k(s,a) < \zeta$ are considered unknown, and the REX-D algorithm will proceed to explore them.

III. Proposed Approach to Avoid Dead-ends

Introducing relational generalizations and LfD has the advantage of reducing the exploration while still permitting to learn successfully different scenarios [4]. Although relational generalizations greatly improve performance [5], they also have drawbacks. Using a relational count function implies that not all states are explored before considering them as known, since all states within a context are assumed to behave likewise. Therefore, state-action pairs that could be needed to attain the best policy might not be visited, and thus their contexts will not be learned. This lack of exploration may lead to models yielding suboptimal policies that fall frequently into dead-ends.

In this section we propose a new method that allows the decision maker to refine the parts of the model that lead to dead-ends, thus producing better policies that avoid those dead-ends while not increasing much the number of exploration actions and teacher demonstrations required. First we explain how dangerous rules (which are rules that may lead to dead-ends) are detected. Then we describe how the decision maker analyzes plans to check whether they include dangerous rules that can lead to a dead end. In this case, the decision maker issues an interaction signal and the teacher, whenever possible, shows alternative actions.

A. Detecting Dangerous Rules

The first requirement to avoid dead-ends is the ability to detect the causes. When the robot cannot find a plan to reach the goal, REX-D issues a teacher demonstration request. When the problem is that the model is incomplete, the teacher shows a new action and the robot continues with the task execution. Contrarily, when the problem is due to a dead-end, there are no further actions that can be applied. In this case, the teacher confirms the dead-end and the procedure to analyze its causes is triggered.

In the proposed model, the dead-end has to be caused by one or more predicates that, when present, prevent the planner from finding a solution. We propose to find these predicates using the so called *excuses* [10]. Excuses are defined as changes to the state that turn the task into a solvable one. In a more formal manner, given an unsolvable planning task involving a set of objects C_π and an initial state s_0, an excuse is a pair $\varphi = \langle C_\varphi, s_\varphi \rangle$ that makes the task solvable, where C_φ is a new set of objects and s_φ a new initial state. Although many valid excuse candidates may exist, we are interested in those that make minimal changes and are not explained by other excuses. Standard planners can be used to find them by adding new actions that include the different candidates, and letting the planner find the best one with the minimal cost, as described in [10].

These excuses are then used to identify the dangerous rules. First we extract the changes in the state that are introduced by the excuse, $\tau_\varphi = s_0 \triangle s_\varphi$ (where $\triangle$ denotes the symmetric set difference). Then we look for rules r whose effects contain a predicate $q \in \Omega_r$ such that $q \in \tau_\varphi$, and we mark such rules as dangerous.

B. Avoiding Dangerous Rules

Algorithm 1 Avoid dangerous rules

```
Input: Dangerous rule $r_d$, current state $s$
 1: $p_{dead} = 0$    ▷ Probability of reaching a dead-end
 2: for All effects $\Omega_{r_d}$ of rule $r_d$ do
 3:     $s' = s$
 4:     Add predicates in $\Omega_{r_d}$ to $s'$
 5:     Plan($s'$)
 6:     if dead-end then
 7:         $p_{dead} += p_{r_d,\Omega_{r_d}}$
 8:     end if
 9: end for
10: if $p_{dead} > \sigma_{r_d}$ then
11:     Request_teacher_confirmation
12: end if
```

Once the dangerous rules have been detected, whenever a new action is planned, the decision maker will check if the state-action pair corresponds to a dangerous rule. In this case a special procedure is executed to analyze the action safety.

Algorithm 1 summarizes the procedure to detect cases where there is danger of falling into a dead-end. The possible effects Ω_{r_d} of the planned dangerous rule r_d are simulated, and the planner is used to check if any of them may lead to an irrecoverable dead-end. After this analysis, the decision maker will know the expected dead-end probability p_{dead}. For each dangerous rule we will have an acceptable risk probability σ_{r_d} that is initialized to 0. If $p_{dead} > \sigma_{r_d}$ then the decision maker will request a confirmation from the teacher before executing the involved action.

The confirmation consists in notifying the teacher about the action that the robot intends to execute, and the teacher will either confirm that it is safe or will demonstrate a different and safer action instead. To handle domains where dead-ends are not completely avoidable, if the teacher confirms a rule as safe the rule's safety threshold will be updated $\sigma_{r_d} = max(\sigma_{r_d}, p_{dead} + \epsilon)$, considering that a risk of p_{dead} is acceptable for that rule. However, if the action is really dangerous, a safer action will be presumably demonstrated by the teacher. This new action will be learned and added to the model, so that the planner will have the option to choose it afterwards.

Another detail that has to be taken into account is that unexplored dangerous rules are also a problem. A RL algorithm will try to explore actions until they are considered known. Therefore, we are considering all dangerous rules as known to avoid selecting them for exploration, as executing them in unknown states may have a high chance of falling into the same dead-ends again.

IV. Experimental Results

Two experiments are proposed to validate our proposal. The first one is a simulated problem from the Triangle Tireworld domain of the 2008 International Probabilistic Planning Competition (IPPC). The second domain is based on the Cranfield benchmark, a standard assembly task that has been implemented in a real robot setting.

A. IPPC: Triangle Tireworld

In the Triangle Tireworld domain a car has to move to its destination, but it has a probability of getting a flat tire while it moves. The car starts with no spare tires but can pick them up in some locations. Safe and long paths exist with spare tires, while the shortest paths don't have any spare tires. We are solving the first problem number of the IPPC 2008 with a probability $p = 35\%$ of getting a flat tire. The actions available in this domain are: a "Move" action to go to an adjacent position, a "Change Tire" action to change a flat tire with a spare tire, and a "Load Tire" action to load a spare tire into the car if there are any in the current location. The main difficulty in the Triangle Tireworld domain are the dead-ends when the agent gets a flat tire and has no spare tires available. If the robot doesn't get a flat tire while it explores locations where spare tires are available, it won't learn how to change a tire and therefore it will always choose the shortest path to the goal where no spare tires are available. It is a challenging domain where RL approaches with a low exploration threshold ζ can fall easily into recurrent dead-ends.

However, the inclusion of dangerous actions permits solving this problem without requiring a large exploration threshold and thus not increasing the number of exploration actions needed to learn the domain. After the robot realizes that moving may lead to spare tires, it will confirm the moving actions with the teacher, who will recommend the safer paths with spare tires instead of the shorter ones. Once the car gets a flat tire with a spare tire available, it will learn how to change tires and perform successfully afterwards. Figure 2 shows the advantages of recognizing and dealing with dangerous actions, where a few extra teacher demonstration requests allow the robot to have a much better success ratio, getting a 98% success ratio after 15 episodes, compared with 88% that would have been obtained without considering dangerous actions (Fig. 2 Left). The cost of this improvement is very low, as just one extra teacher demonstration is needed in average (Fig. 2 Right).

B. Cranfield benchmark

This experiment was performed in the context of the IntellAct EU project using the MARVIN platform, which integrates the decision maker presented in this paper. The MARVIN platform has a set of perception modules, including object recognition, tracking [14], semantic event chains [15], predicate estimation and manipulation monitoring to provide the decision maker with all the input required to reason about the best actions to be performed. On the execution side, DMPs are used to learn demonstrated actions [16]. An overview of the system, including a description of all the modules involved is presented in [17]. The important details to be considered for this work are that the decision maker receives a list of symbolic predicates representing the current state as input, and that actions executed by the human teacher are also recognized. Finally, to execute an action, the decision maker has to provide the robot with the name of the action and its parameters.

The Cranfield benchmark consists in assembling an industrial piece, whose parts are shown in Fig. 3. It is a standard assembly task in robotics. There are precedence constraints in the assembly that restrict the order in which the parts have to be assembled. Square pegs have to be placed before the separator, the shaft before the pendulum, and all other pieces before the front faceplate.

The problem to be solved consisted of two parts. First, the robot had to complete the standard Cranfield benchmark scenario during the first 10 episodes, after which the initial setup was changed. From the $11th$ episode onwards, a peg was laying down horizontally on the table in the initial state. The robot action to place a peg is more difficult when the peg is laying horizontally, and it may fall and roll away if not grasped properly. However, there is an action that repositions the peg in a vertical position which permits safer grasps.

This action has to be learned with a new teacher demonstration when a peg is laying horizontally, as it isn't needed during the first 10 episodes. In the proposed system, the robot actively requests demonstrations from the teacher. Thus, in this case, the robot has to realize that it needs a new action and request it. When not recognizing dangerous actions, the robot will try to place the pegs directly from the horizontal position because this action has some probability of performing well. In contrast, a place peg action with an horizontal peg can be recognized as a dangerous action, and therefore the action would have to be confirmed by the

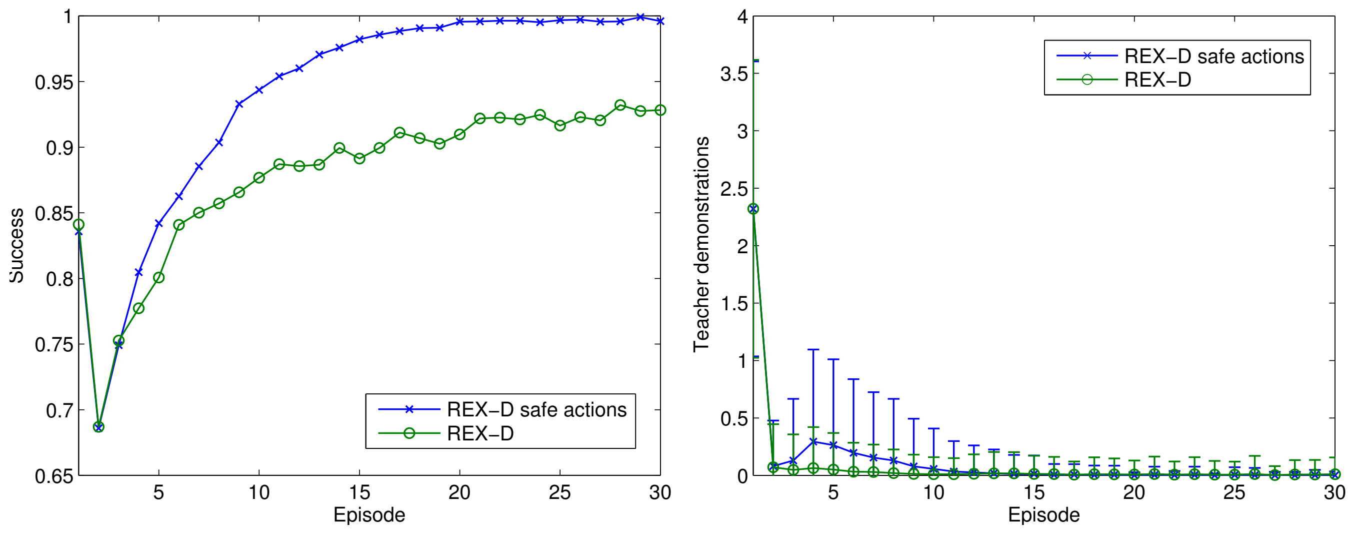


Fig. 2. Results in the Triangle Tireworld domain. The mean and standard deviations obtained over 2500 runs are shown. The exploration threshold is $\zeta = 2$. **Left:** the success ratio of the algorithms. **Right:** the number of teacher demonstrations requested.

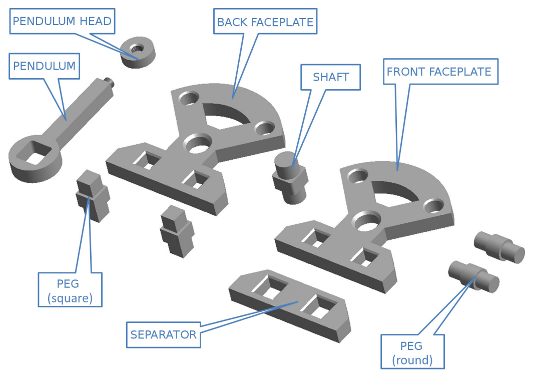


Fig. 3. Cranfield benchmark assembly. To assemble it, the different parts have to be placed in a restricted order: the separator requires the square pegs to be placed, the pendulum must be placed after the shaft, and the front faceplate requires all pegs, the separator and the pendulum to be in place.

teacher, who would suggest to reposition the peg vertically instead.

The MARVIN robot (Fig. 1) was used to assemble the Cranfield benchmark and validate the algorithm. Videos of the robot working on the Cranfield benchmark can be seen in `http://www.iri.upc.edu/groups/perception/safePolicies`. Unfortunately, no systematic experiments were carried out that permit obtaining meaningful statistics about its performance. To get statistical results we executed the experiment in a simulator, which allowed us to repeat it 500 times and obtain more accurate results.

Figure 4 shows the results obtained. A few episodes after introducing horizontal pegs, confirming which actions are dangerous improves significantly the success ratio, getting over 90% in 11 episodes (Fig. 4 Left), while requiring for that improvement just an average of 0.9 extra teacher demonstrations (Fig. 4 Right).

V. Conclusions

We have presented a method to learn safe policies within model-based reinforcement learning algorithms with very sparse exploration. The robot is able to reason about dead-ends and analyze the causes, to later confirm with the teacher which actions were dangerous before executing them again and falling into the same dead-ends. In addition to learning how to avoid dead-ends, the robot is the one actively monitoring the scenario and interacting with the teacher when needed, releasing the teacher from monitoring the robot continuously. Experimental results have proven the advantages of the method, both in an IPPC domain and a robotics scenario. Our approach was shown to reach success ratios close to 1 with just a few extra demonstration requests and exploration actions.

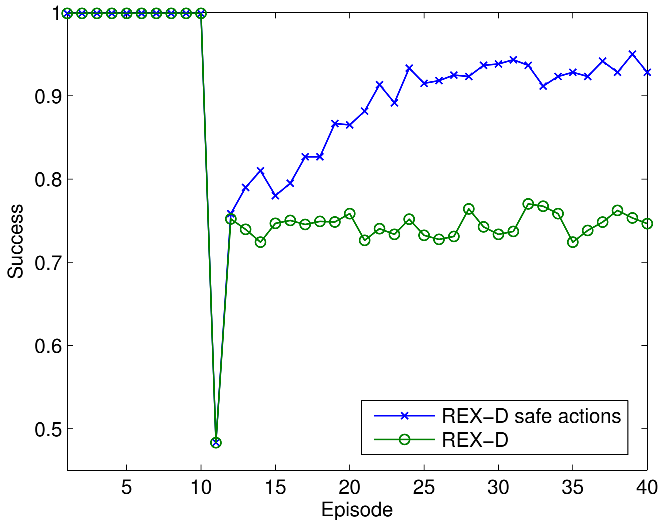
As future work, we would like to extend the approach to work under partial observability, which is an ubiquitous problem in robotics. This would require to improve the analysis of dead-ends to consider candidates that may not have been observed, and integrate a POMDP planner that could tackle partial observations.

Acknowledgment

Thanks to Thiusius Rajeeth Savarimuthu and Norbert Krüger for their help with the MARVIN platform, as well as the whole IntellAct EU project consortium that integrated the different skills in the robot to enable it to assemble the Cranfield benchmark used in the experiments.

References

[1] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, pp. 579–610, 2012.
[2] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2003.
[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.



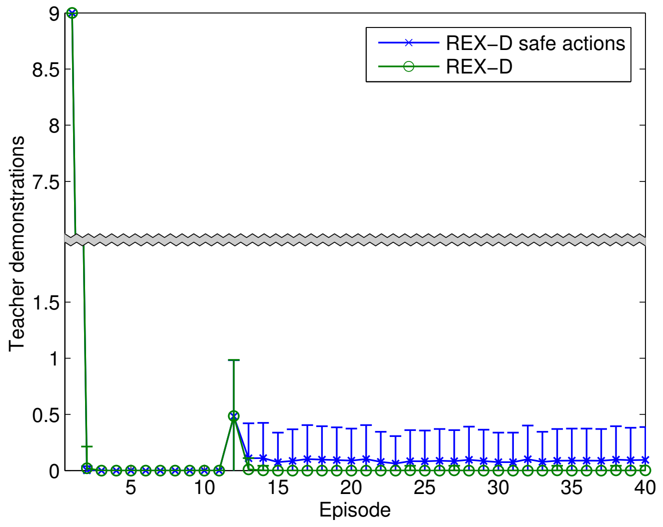


Fig. 4. Results in the Cranfield benchmark scenario. The mean and standard deviations obtained over 500 runs are shown. The exploration threshold is $\zeta = 2$. During the 10 first episodes, all pegs start in a standard vertical position, and after the 11^{th} episode a peg is always positioned laying horizontally on the table. **Left:** the success ratio of the algorithms. **Right:** the number of teacher demonstrations requested.

[4] D. Martínez, G. Alenyà, P. Jiménez, C. Torras, J. Rossmann, N. Wantia, E. E. Aksoy, S. Haller, and J. Piater, "Active learning of manipulation sequences," in *Proc. of International Conference on Robotics and Automation*, 2014, pp. 5671–5678.

[5] T. Lang, M. Toussaint, and K. Kersting, "Exploration in relational domains for model-based reinforcement learning," *Journal of Machine Learning Research*, vol. 13, pp. 3691–3734, 2012.

[6] F. De Jonge, N. Roos, and C. Witteveen, "Primary and secondary diagnosis of multi-agent plan execution," *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 2, pp. 267–294, 2009.

[7] R. Van Der Krogt and M. De Weerdt, "Plan repair as an extension of planning." in *Proc. of the International Conference on Automated Planning and Scheduling*, vol. 5, 2005, pp. 161–170.

[8] M. Gianni, P. Papadakis, F. Pirri, M. Liu, F. Pomerleau, F. Colas, K. Zimmermann, T. Svoboda, T. Petricek, G.-J. M. Kruijff, *et al.*, "A unified framework for planning and execution-monitoring of mobile robots." in *Proc. of the AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, 2011.

[9] S. Karapinar, S. Sariel-Talay, P. Yildiz, and M. Ersen, "Learning guided planning for robust task execution in cognitive robotics," in *Proc. of the AAAI Workshop on Intelligent Robotic Systems*, 2013.

[10] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel, "Coming up with good excuses: What to do when no plan can be found." in *Proc. of the International Conference on Automated Planning and Scheduling*, 2010, pp. 81–88.

[11] M. V. Menezes, L. N. de Barros, and S. do Lago Pereira, "Planning task validation," in *Proc. of the ICAPS Workshop on Scheduling and Planning Applications*, 2012, pp. 48–55.

[12] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, no. 1, pp. 309–352, 2007.

[13] A. Kolobov, P. Dai, M. Mausam, and D. S. Weld, "Reverse iterative deepening for finite-horizon mdps with large branching factors," in *Proc. of the International Conference on Automated Planning and Scheduling*, 2012.

[14] J. Papon, A. Abramov, and F. Wörgötter, "Occlusion handling in video segmentation via predictive feedback," in *Proc. of the ECCV Workshops and Demonstrations*, 2012, pp. 233–242.

[15] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of object–action relations by observation," *International Journal of Robotics Research*, vol. 30, no. 10, pp. 1229–1249, 2011.

[16] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[17] T. R. Savarimuthu, A. G. Buch, Y. Yang, S. Haller, J. Papon, D. Martínez, and E. E. Aksoy, "Manipulation monitoring and robot intervention in complex manipulation sequences," in *Proc. of the RSS Workshop on Robotic Monitoring*, 2014.